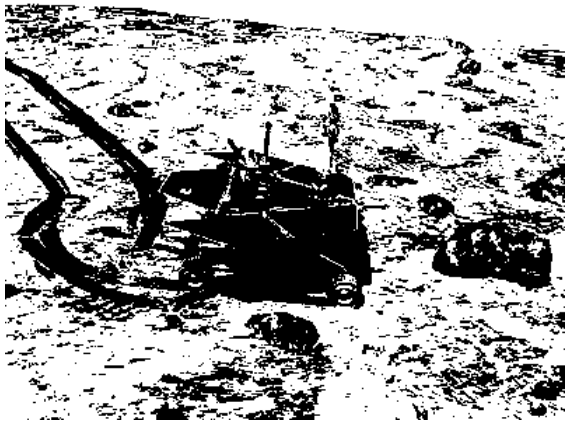
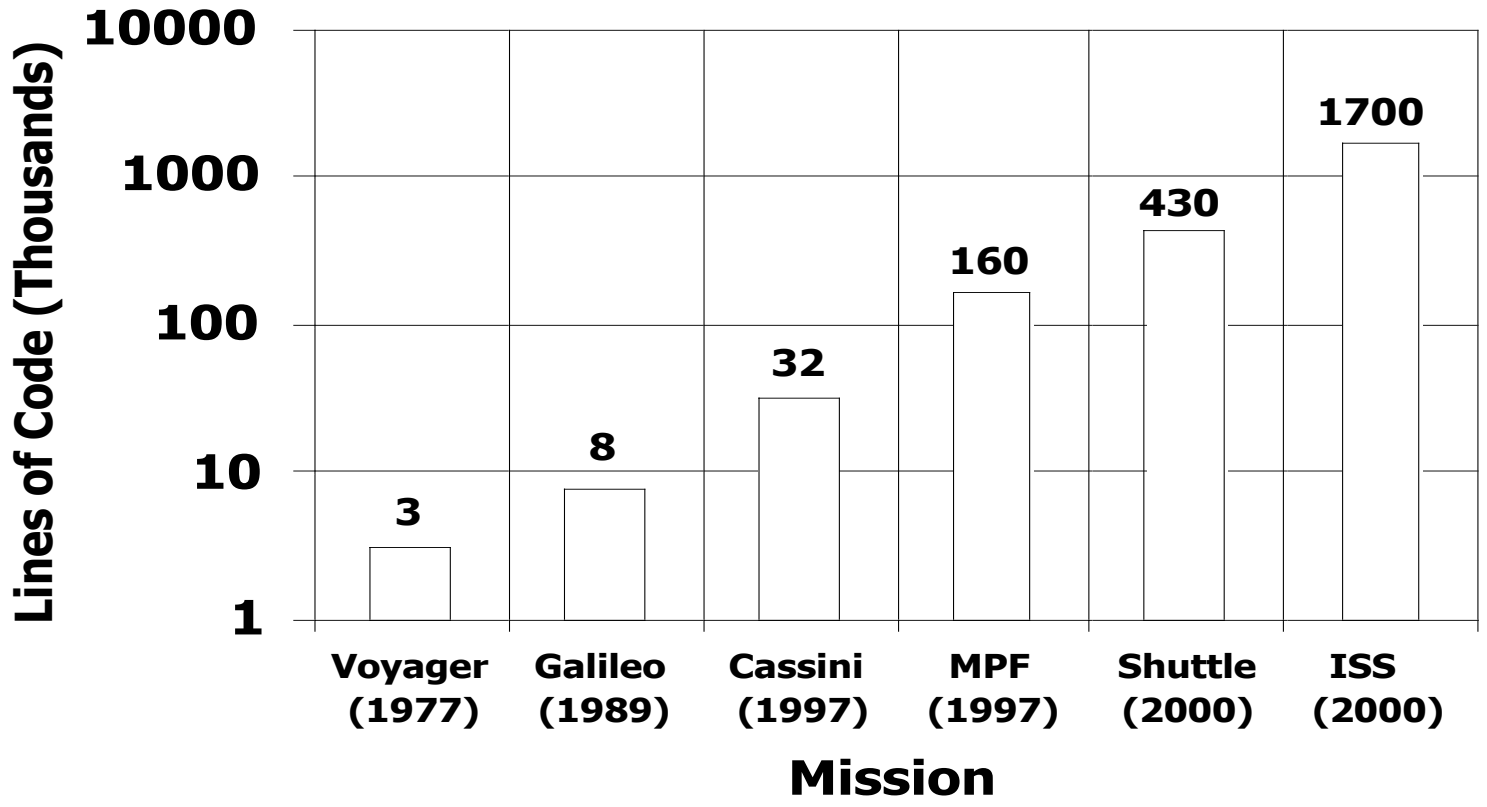
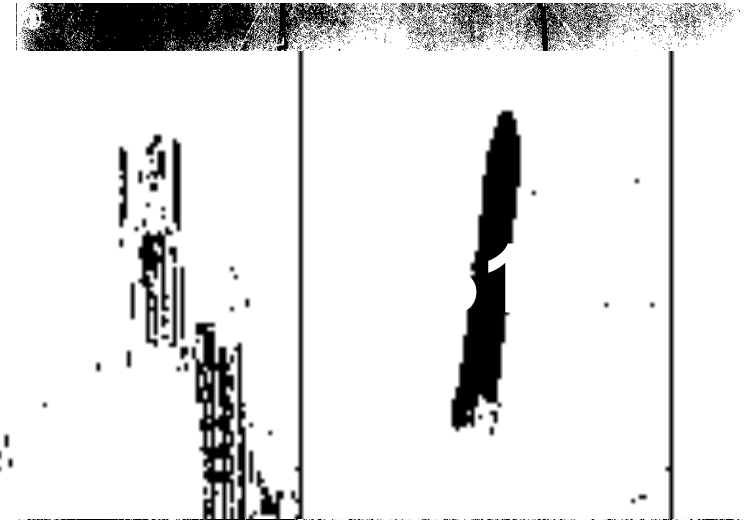
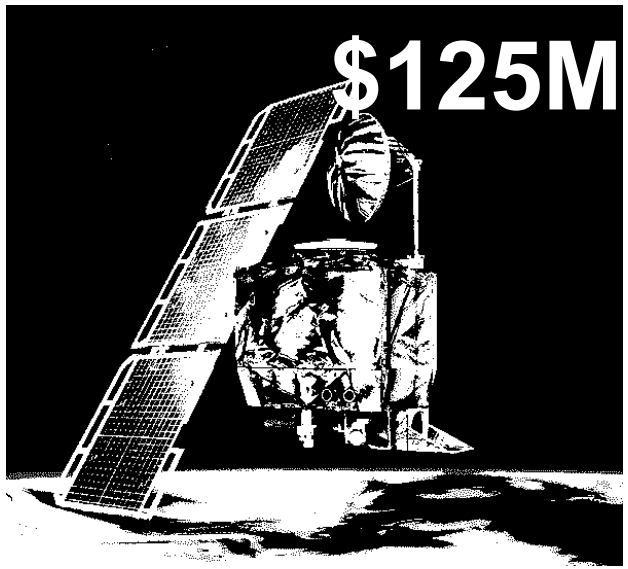


Guillaume Brat

USRA/RIACS


Lines of Code (Thousands)
10000
1000
100
10
1
3
8
32
160
430
1700
Voyager (1977)
Galileo (1989)
Cassini (1997)
MPF (1997)
Shuttle (2000)
ISS (2000)
Mission




$125M


4 months
lost

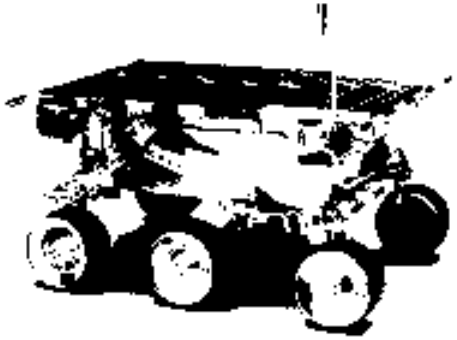
- Need to develop three systems for each mission:
  - Flight software
  - Ground software
  - Simulation software
- Flight software
  - Has to fit on radiation-hardened processors
  - Limited memory resources
  - Has to provide enough information for diagnosis
  - Can be patched (or uploaded) during the mission
- Each mission has its own goals, and therefore, each software system is unique!
- Cannot benefit from opening its source code to the public because of security reasons.
  - No open-source V&V
- Mission software is getting more complex.
  - Large source code (~1 MLOC)
  - The structure of the code is more complex

- International Space Station:
  - Attitude control system, 1553 bus, science payloads
  - International development (interface issues)
  - Codes ranging from 10-50 KLOC
  - A failure in a non critical system can cause a hazardous situation endangering the whole station
  - Enormous maintenance costs

- SCR 25345 describes an issue where GNC Redundancy Management (RM) does not appropriately reset “Indicate Attitude Control Handover to RS" Flag .
  - Flag set (4 occurrences since Feb’03 CCS R3 uplink)
    - On GNC MDM failure
    - SMTC loss of communication (triggers GNC failure response)
    - Planned GNC MDM swaps
  - If flag set, Autohandover to RS Enabled, RS is in Mode of CMG TA or Indicator, and US is Master; FDIR will send an Off Nominal US to RS H/O command.
  - If this flag is not reset an attitude control force fight will occur.

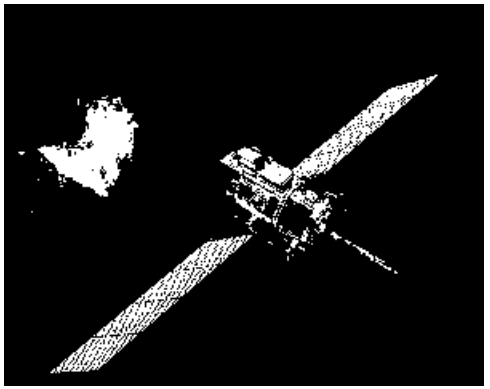
Dan Duncavage, NASA JSC, June 2003

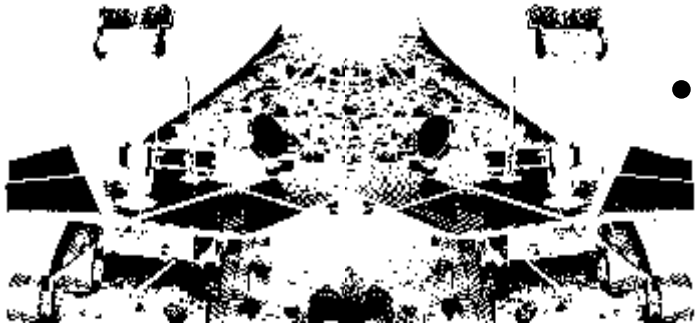
“Are these problems that ANY sort of computational assistance will help? I always knew that we couldn't build a complete system that would automatically tell us what problems would occur with this or that software change. But I am hoping that we can build tools that make things a whole lot faster than they are now. “



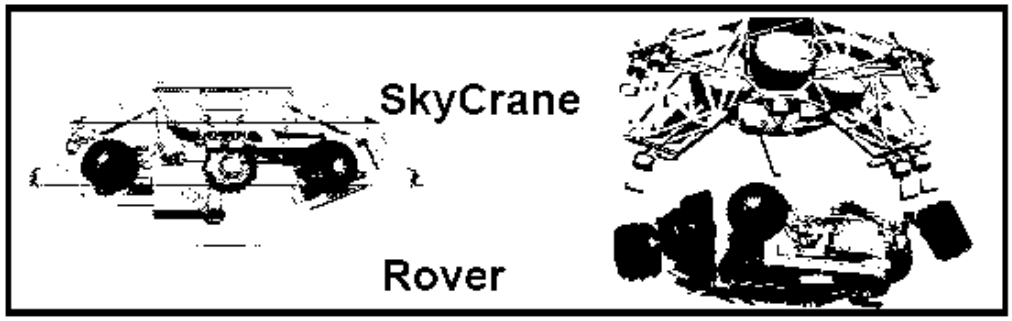
- Mars Path Finder:
  - Code size: 140 KLOC
  - Famous bug: priority inversion problem

- Deep Space One:
  - Code size: 280 KLOC
  - Famous bug: race condition problem in the RAX software

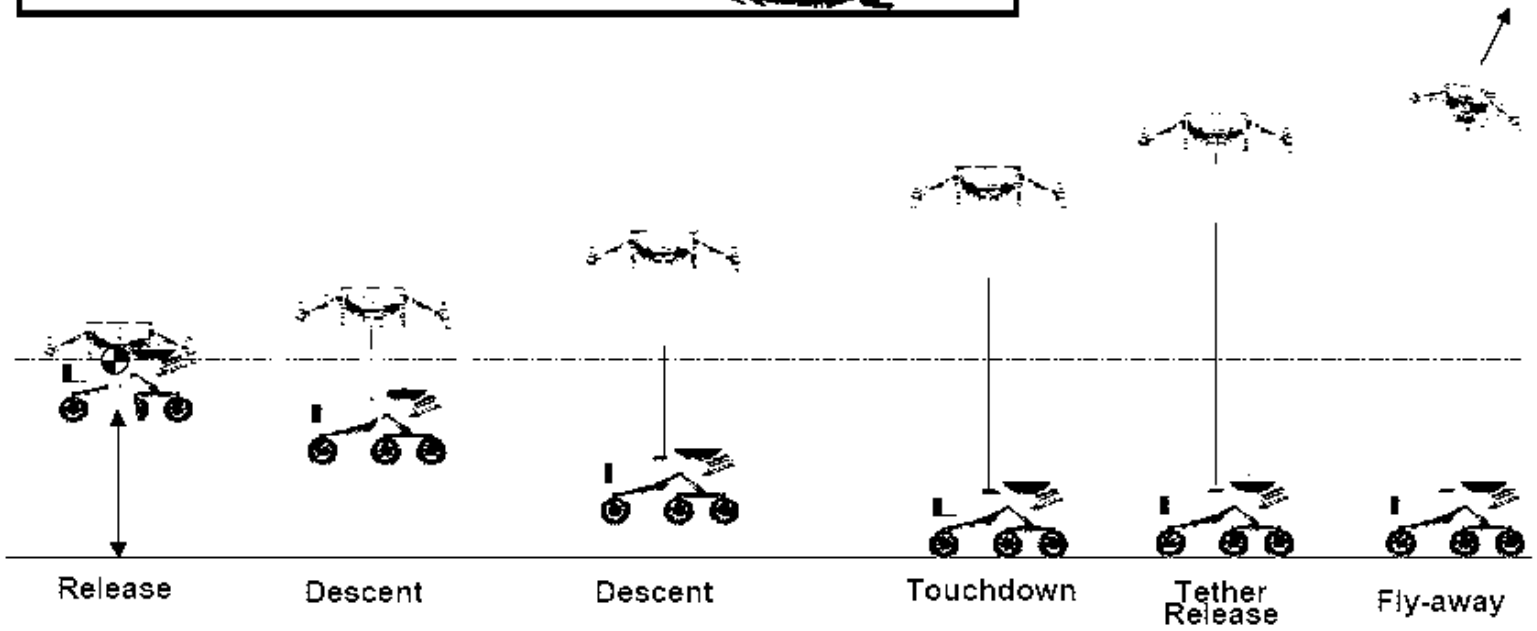




- Mars Exploration Rovers:
  - Code size: > 650 KLOC
  - Famous bug: Flash memory problem



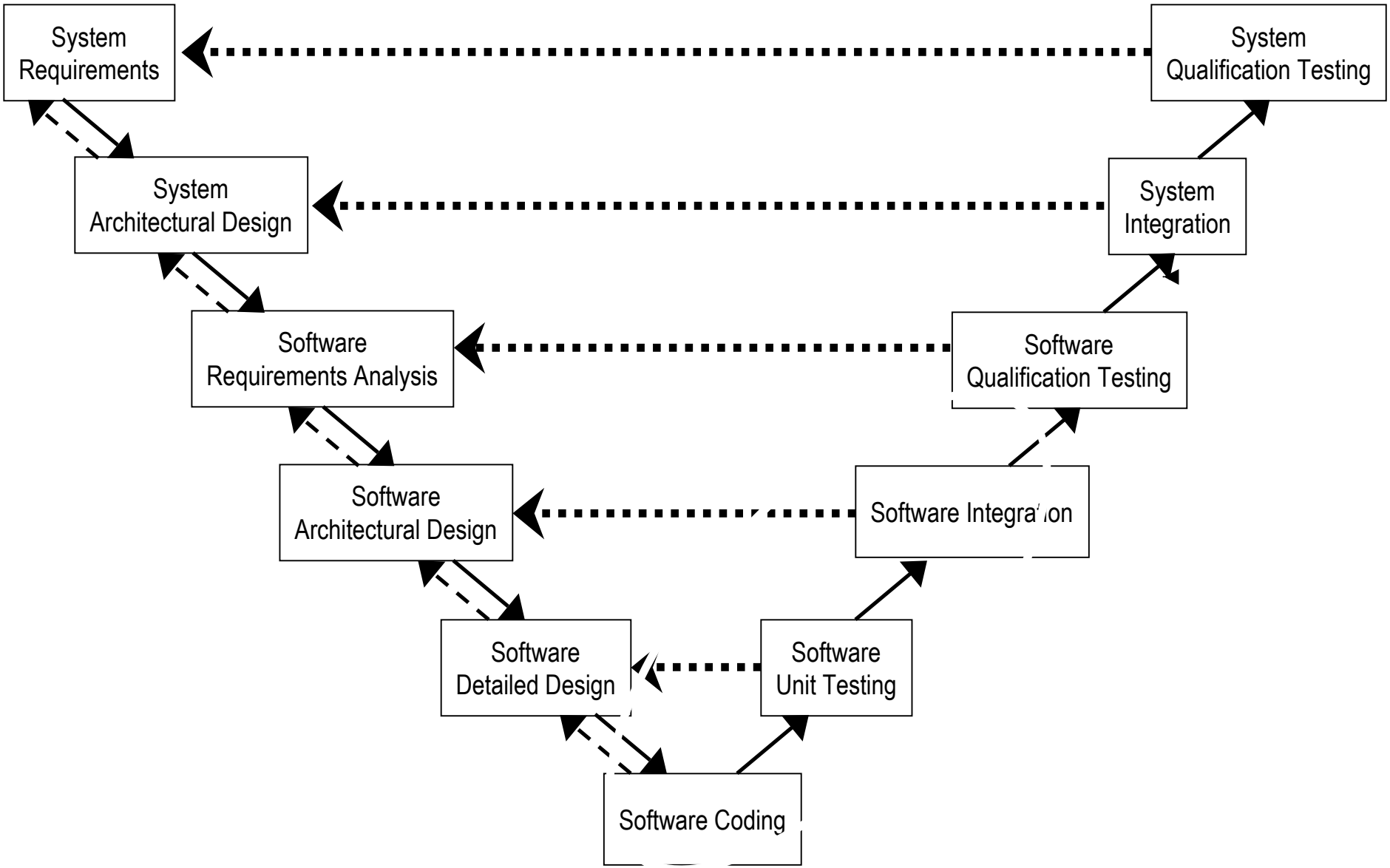
PRE-DECISIONAL DRAFT; For planning and discussion purposes only



- Complicated Landing:
  - no ground real time control
  - The rover lands, the crane flies away
- Long autonomous traverses
  - Automatic obstacle avoidance
  - Recognize possible interesting science along the way
- Critical systems:
  - Uses RTG (no solar panels) for power
- It’s a long mission, almost 2 years of rover operation
  - Needs to be durable
  - Plenty of time to recover in case of problems

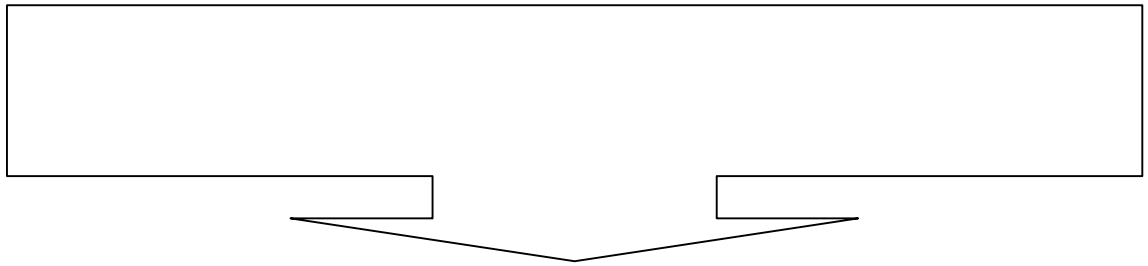
- Mars missions: high-fidelity test bench
  - Runs 24 hours a day
  - 8 hour test sessions:
- Space Station:
  - Critical software: on-ground simulator maintained at Marshall Space Center
  - Payloads:
    - Independently verified by contractors
    - NASA test requirement document

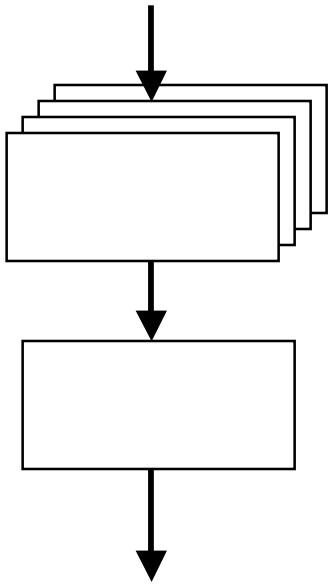
- Badly re-initialized state variable for MPL:
- Unit mismatch for MCO:
- Thread priority inversion problem for MPF:
- Flash memory problem for MER:
- Science mission for the ISS currently under validation:
  - Passes NASA test requirements

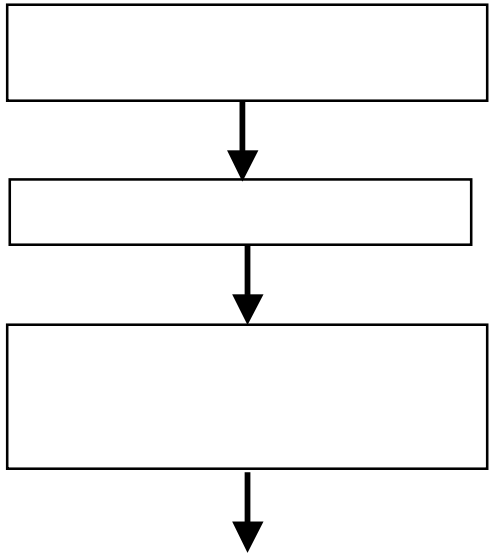

System Requirements
System Qualification Testing
System Architectural Design
System Integration
Software Requirements Analysis
Software Qualification Testing
Software Architectural Design
Software Integration
Software Detailed Design
Software Unit Testing
Software Coding

Static analysis offers compile-time techniques for predicting safe and computable approximations to the set of values arising dynamically at run-time when executing the program

We use abstract interpretation techniques
to extract a safe system of semantic equations
which can be resolved using lattice theory techniques
to obtain numerical invariants for each program point







```
    p =    - 0.75;
    y =       ( );
}

/* unreachable or dead code
void unr () {
   int x = random_int );
   int y = random_int );
       if (  >  ) {
                x =      ;
                if (   < 0) {
```

- Static analysis is well-suited for catching runtime errors
  - Overflow/Underflow
  - Invalid arithmetic operations
- Also for program understanding
  - Data dependences
  - Control dependences
  - Slicing
- Potential applications to
  - Convergence/divergence in floating point computations
  - Unit mismatching
  - Execution time predictions
  - Memory usage predictions

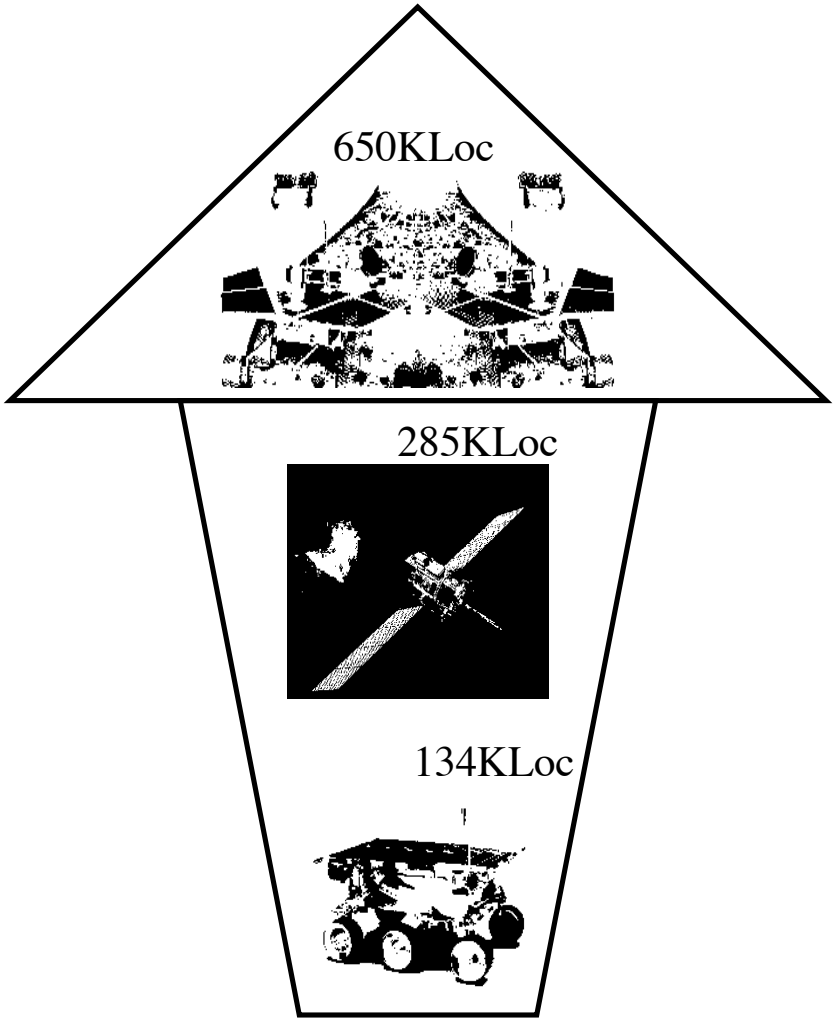
Identification of
technical gaps

Identification of
commercial tools

Experiments on
real NASA code

Implementation of
research prototype

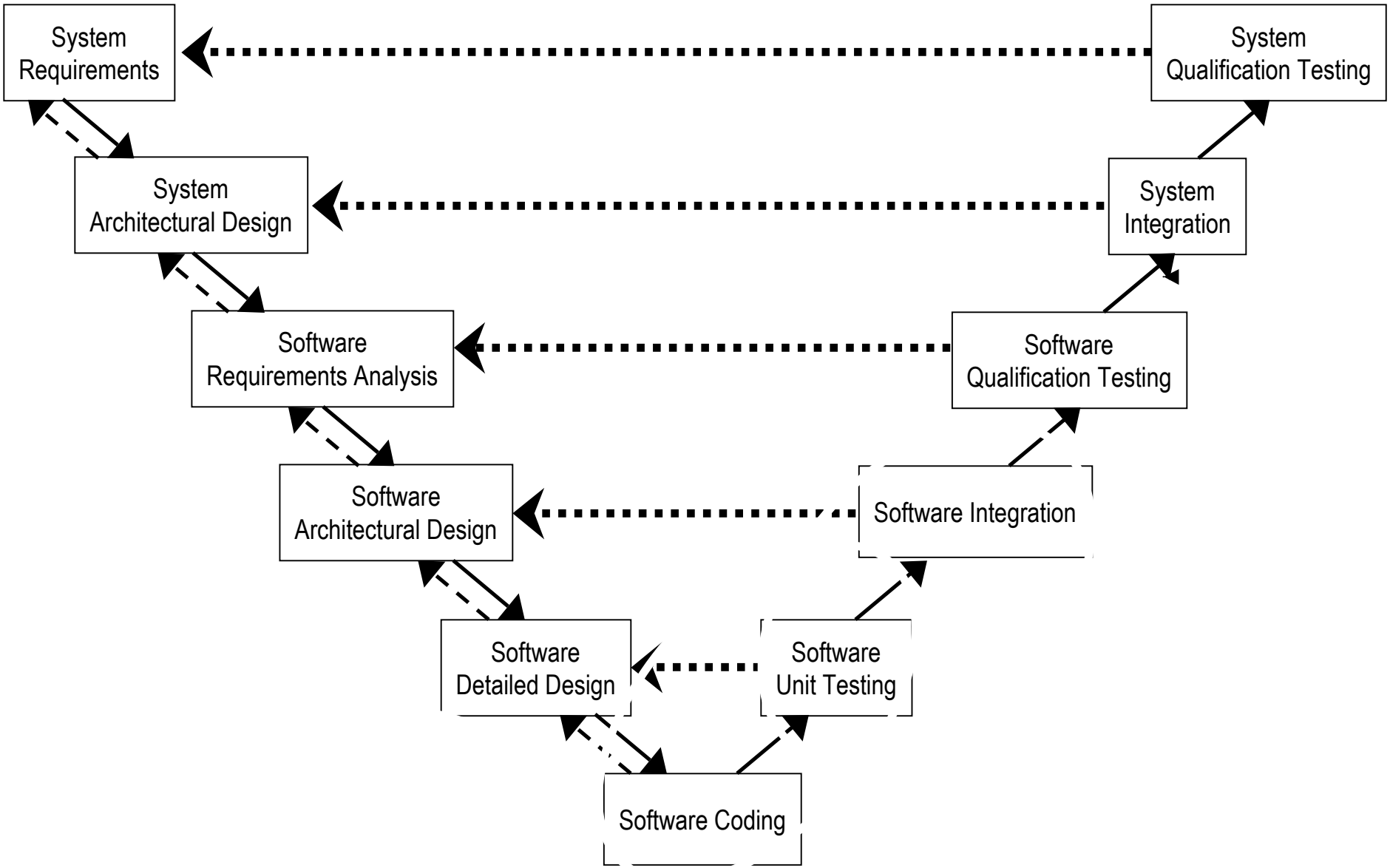
POLYSPACE C-VERIFIER

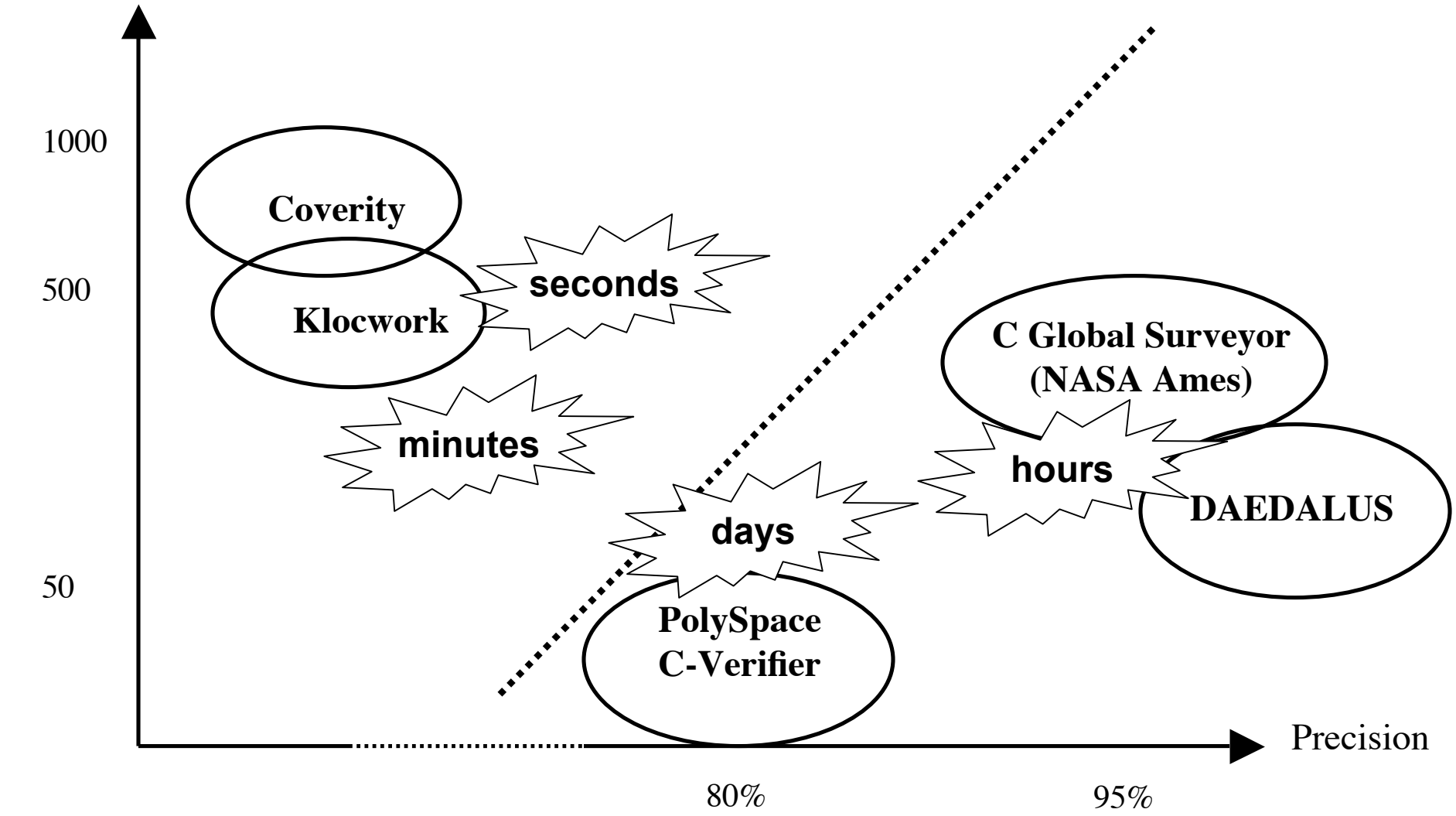


Found errors!

Un-initialized variables
Out-of-bound array accesses
Overflow/underflow problems

- We conducted extensive experiments with PolySpace Verifier:
  - Minors bugs found in MER
  - Serious out-of-bounds array accesses found in an ISS Science Payload
- Useful:
- Effective:
  - It takes 24 hours to analyze 40 KLOC
  - Difficulty to break down large systems into small modules

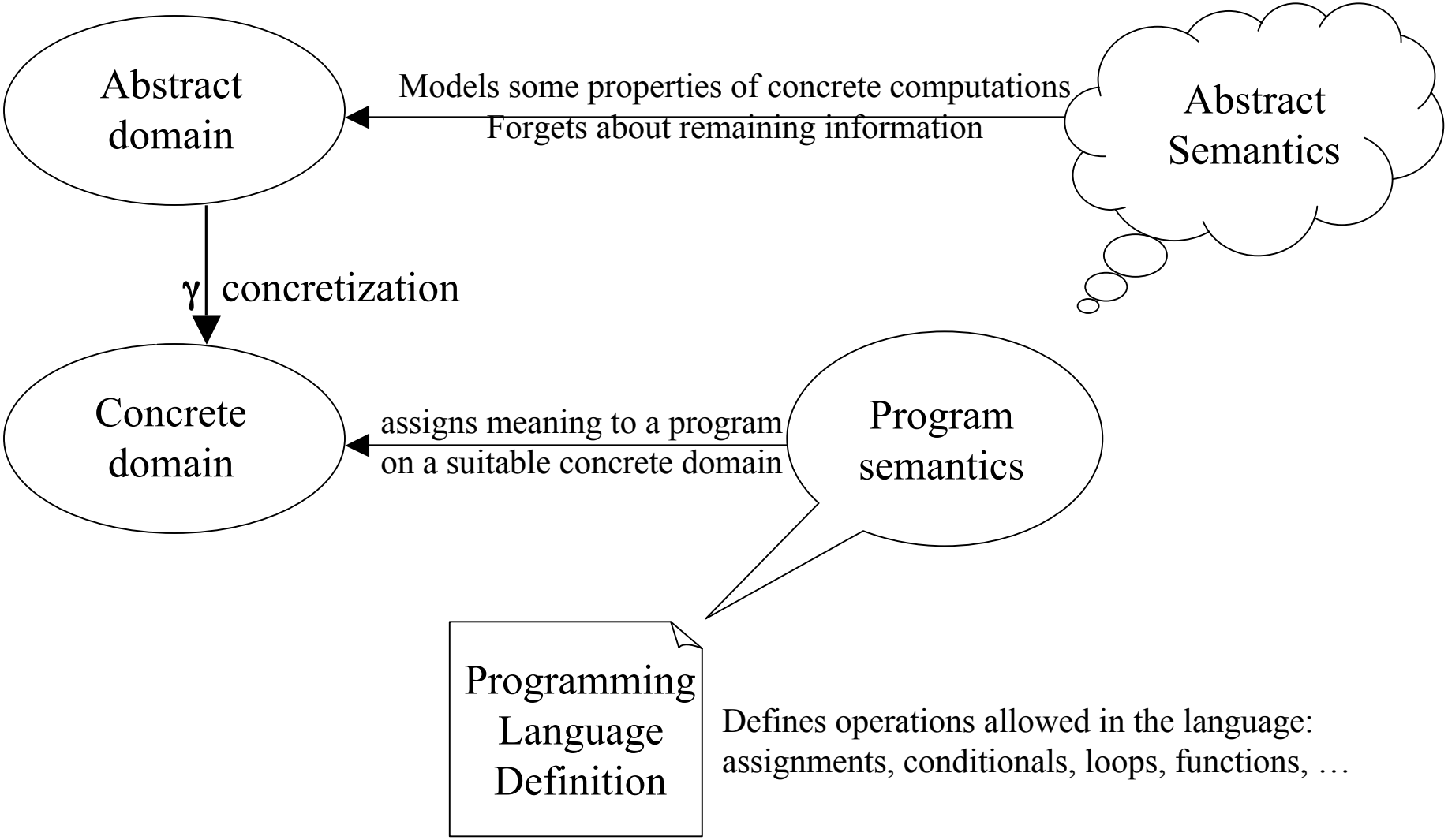

System Requirements
System Qualification Testing
System Architectural Design
System Integration
Software Requirements Analysis
Software Qualification Testing
Software Architectural Design
Software Integration
Software Detailed Design
Software Unit Testing
Software Coding

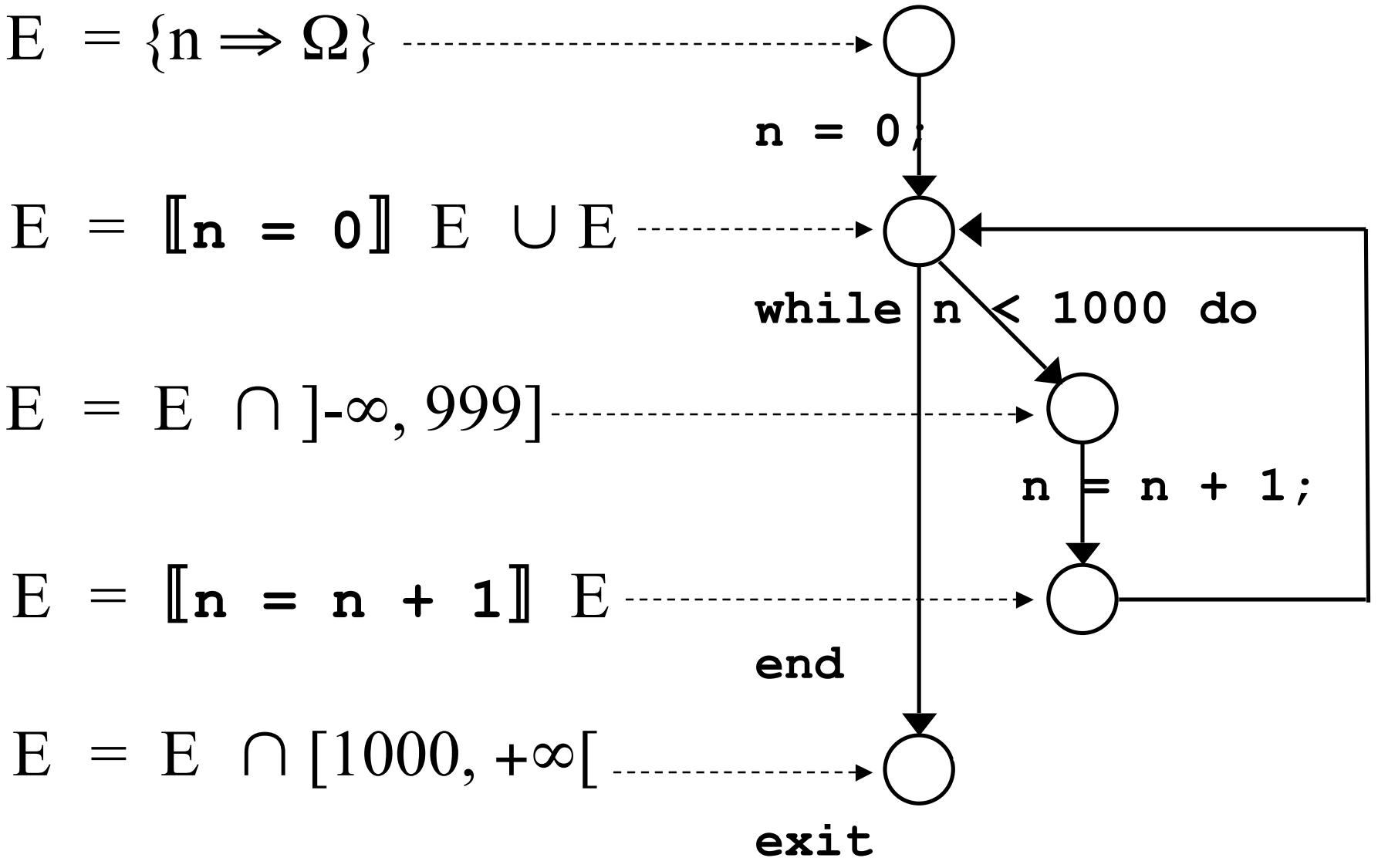

Scalability (KLOC)
1000
500
50
Coverity
Klocwork
seconds
minutes
days
C Global Surveyor
(NASA Ames)
hours
DAEDALUS
PolySpace
C-Verifier
Precision
80%
95%

  - Analyze large systems in less than 24 hours
  - Analysis time similar to compilation time for mid-size programs
- Precision:
  - At least 80%

    the analysis provides enough information to diagnose a warning

- Prototype analyzer
  - Based on abstract interpretation
  - specialized for NASA flight software
- Covers major pointer manipulation errors:
  - Out-of-bounds array indexing
  - Uninitialized pointer access
  - Null pointer access
- Keeps all intermediate results of the analysis in a human readable form:


Abstract domain
Models some properties of concrete computations
Forgets about remaining information
Abstract Semantics
γ concretization
Concrete domain
assigns meaning to a program
on a suitable concrete domain
Program semantics
Programming Language Definition
Defines operations allowed in the language:
assignments, conditionals, loops, functions, …

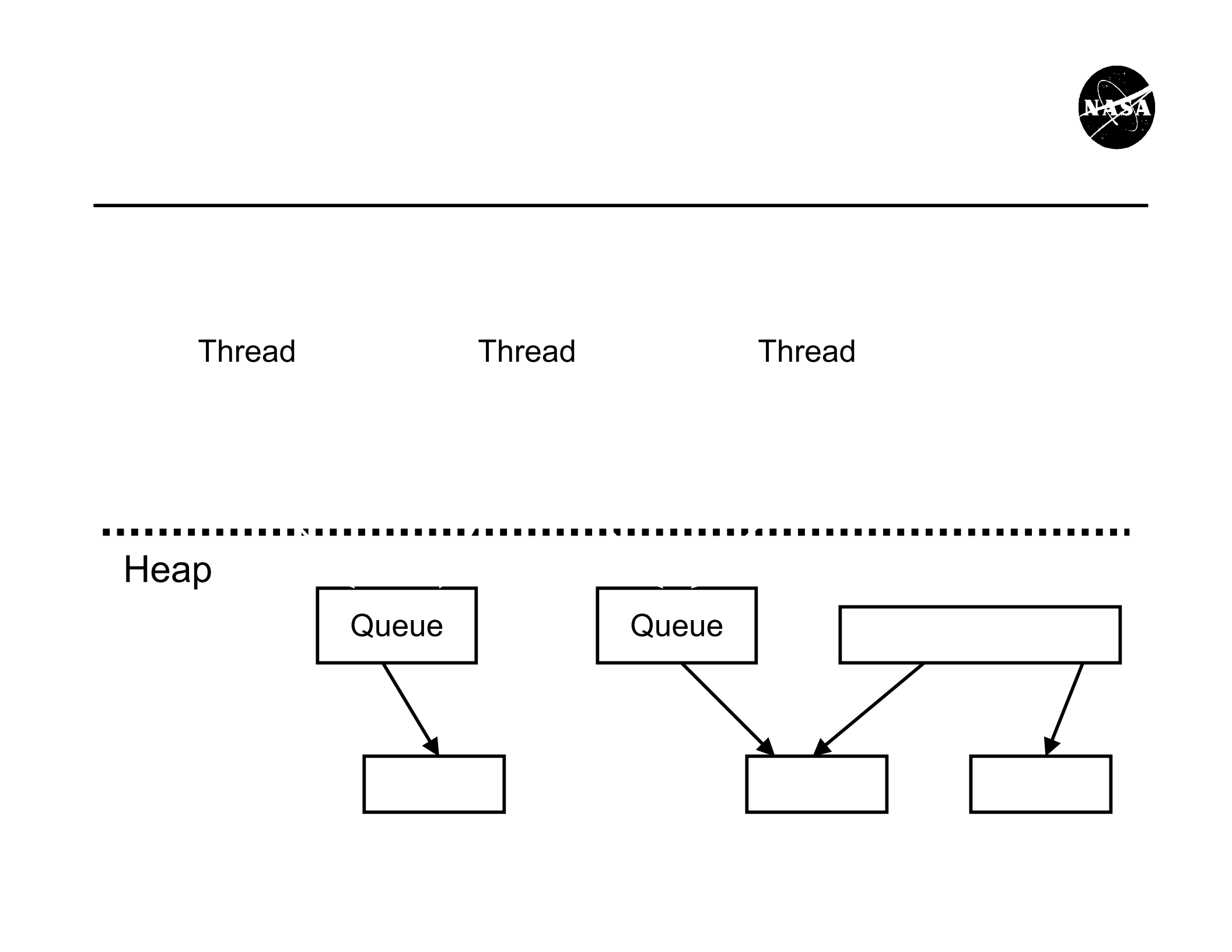

E = {n ⇒ Ω}
n = 0;
E = ⟦n = 0⟧ E ∪ E
while n < 1000 do
E = E ∩]-∞, 999]
n = n + 1;
E = ⟦n = n + 1⟧ E
end
E = E ∩ [1000, +∞[
exit

In effect, the analysis has automatically computed numerical invariants!

```
n = 0;

while n < 1000 do

        n = n + 1;

end

exit
```

- Arrays are the basic data structures in embedded programs
- Out-of-bounds array access
  - One of the most common runtime errors
  - One the most difficult to trace back

```
double a[10];
for (i = 0; i < 10; i++)
                    <-------------  0 <= i < 10
if (...)
                    <-------------  i = 10
```

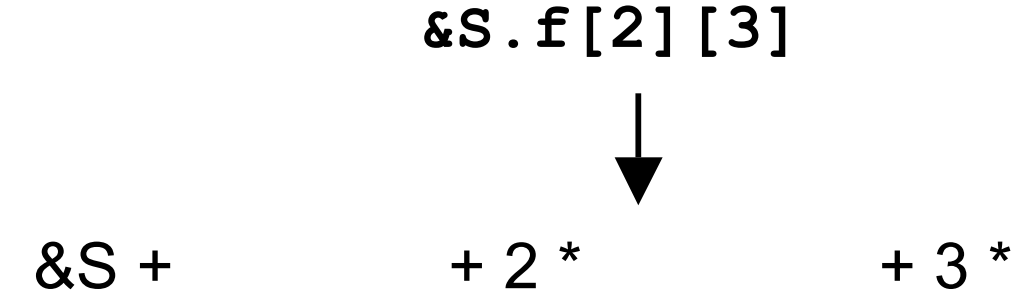

Thread
Thread
Thread
Heap
Queue
Queue


```
assign (double *p, double *q, int n) {
  int i;
  for (i = 0; i < n; i++)
    p[i] = q[i];
}
```

- Context-sensitivity is required
- We can't afford performing 1000 fixpoint iterations with widening and narrowing for each function
- Compute a summary of the function using a relational numerical lattice

```
access(p[i],                    )

access(q[i],                    )
```

- Pointer analyses commonly use symbolic access paths into structures
- Mixing symbolic and numerical information is difficult and costly
- We use a uniform byte-based representation (sufficient for array bound checking)



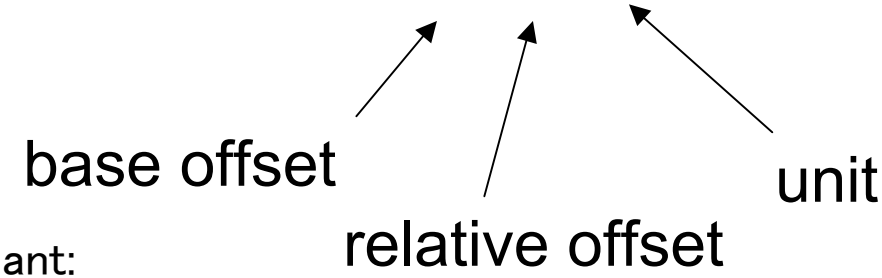
- Convex polyhedra are too costly (exponential complexity)

- Weakly relational domain of Difference-Bound Matrices (Mine 01):
  - $\{x - y \leq c, z - t \leq c', ...\}$
  - Floyd-Warshall algorithm (shortest path):
    - $x - y \leq c \;\&\; y - z \leq c' \Rightarrow x - z \leq c + c'$
    - $x - y \leq c, x - y \leq c' \Rightarrow x - y \leq \min(c, c')$
  - Cubic time, quadratic space complexity

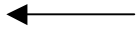
- Cannot express the invariant:

  $0 \leq$ **offset** $\leq$ `n *`

- Solution: use auxiliary variables
  - Split up the offset: **offset = b + δ * u**

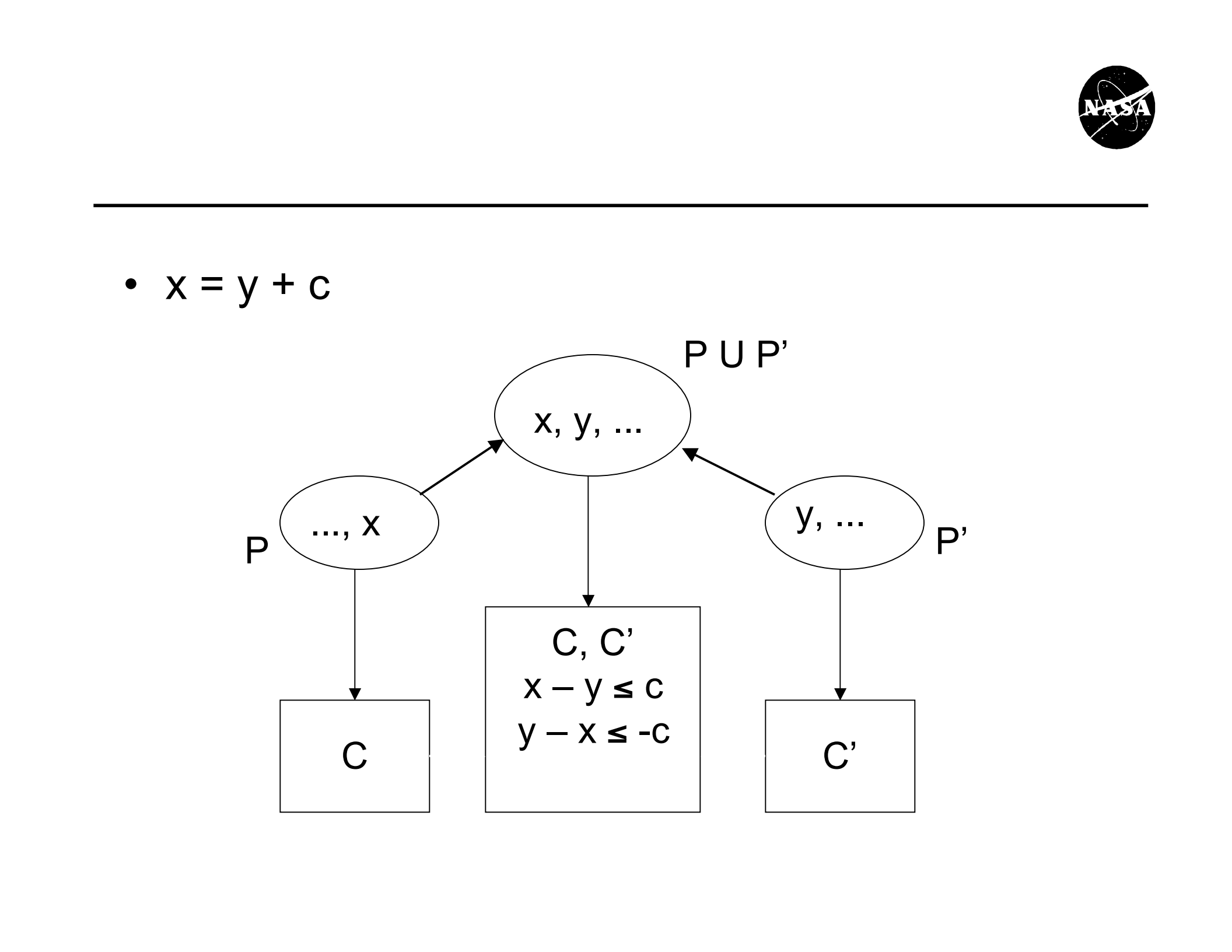


  - New invariant:
    - **b** = 0
    - **u** = `sizeof (double)`
    - `0 ≤ δ ≤ n`



Expressible as a DBM

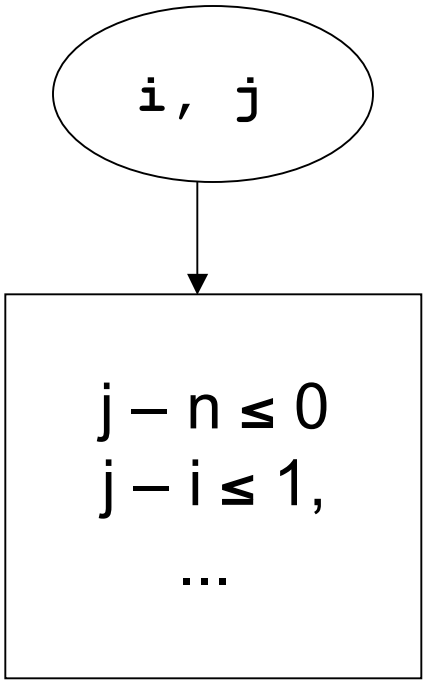
- The domain of Difference Bound Matrices do not scale
- Problem: strongly polynomial (worst-case bounds always attained)
- Solution: split up the relations into small packets using computational dependencies

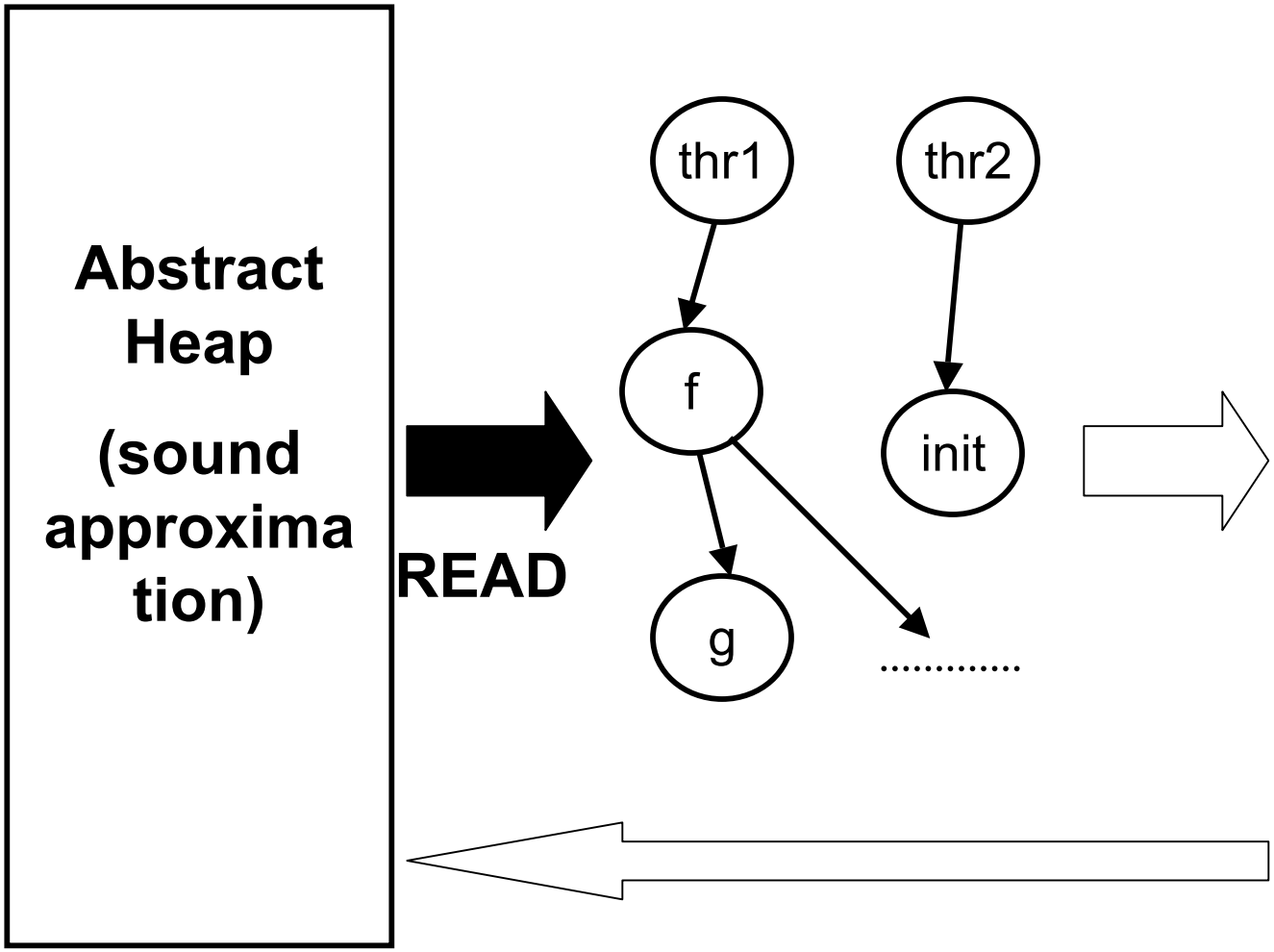


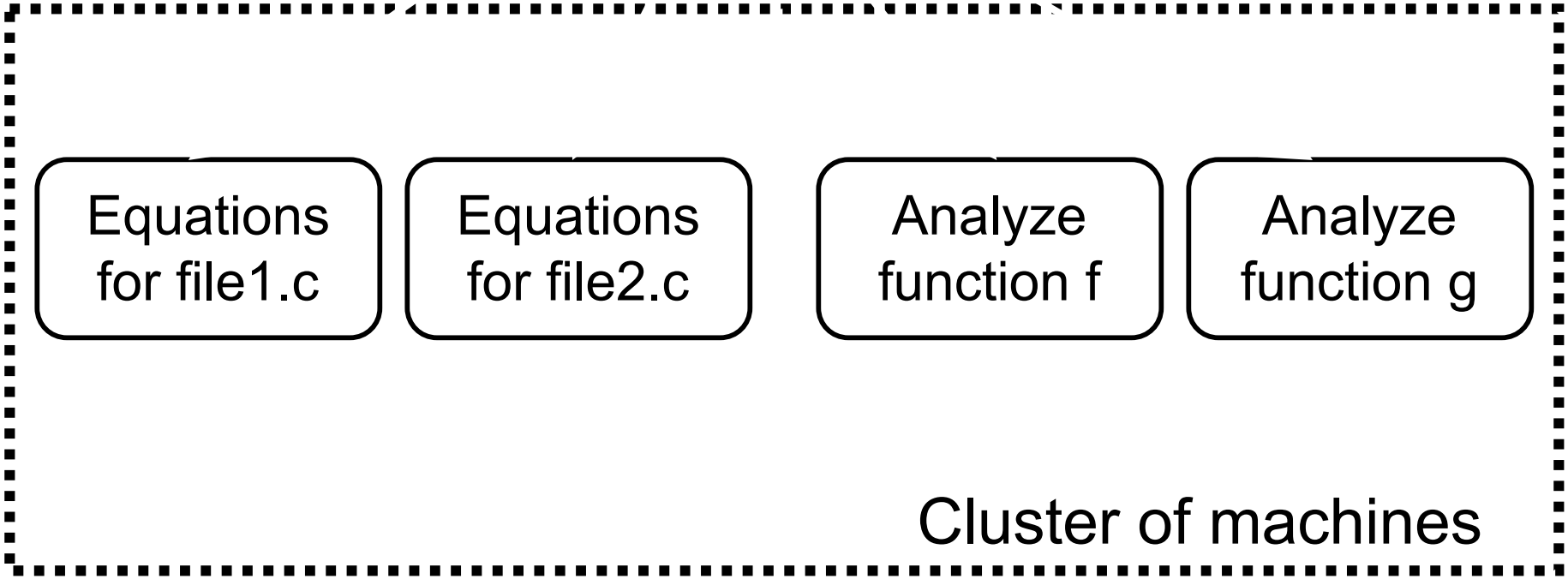
- x = y + c

- All variable modified within a loop are clustered (implicit dependencies)

```
j = 1;
for (i = 0; i < n; i++) {
  j++;
  a[j] = ...;
}
```

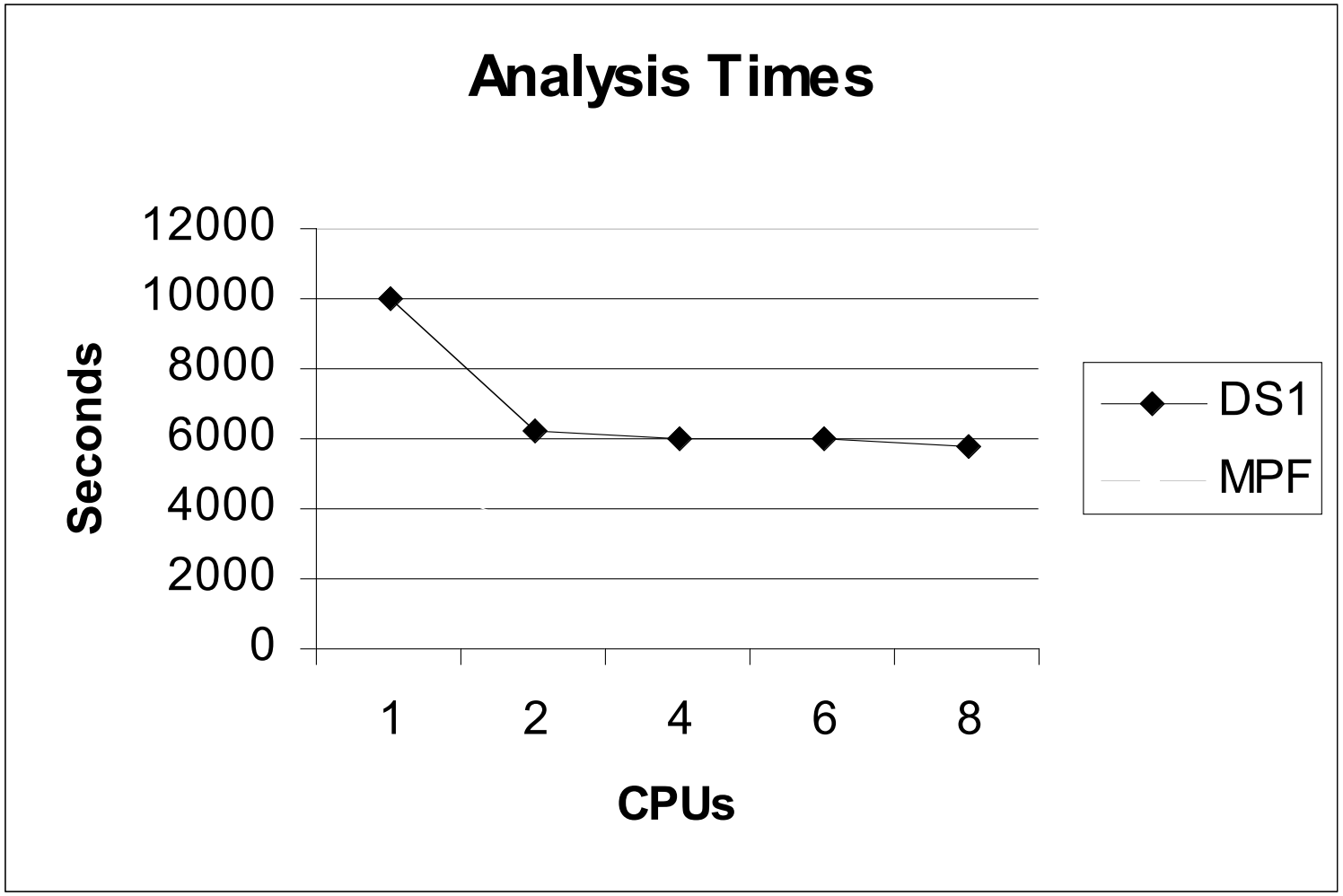



Abstract Heap
(sound approxima tion)
READ
thr1
thr2
f
init
g
.....


Equations for file1.c
Equations for file2.c
Analyze function f
Analyze function g
Cluster of machines

- We use PostgreSQL
- Mutual exclusion problems are cared for by the database
- Simple interface using SQL queries

- We use the Parallel Virtual Machine (PVM)
- High-level interface for process creation and communication
- Allows heterogeneous implementation: currently a mix of C and OCaml


Analysis Times
12000
10000
8000
6000
4000
2000
0
Seconds
1
2
4
6
8
CPUs
DS1
MPF

- The performance curve flattens: overhead of going through the network
- MER takes a bit less than 24 hours to analyze:
  - 70% of the time is spent in the interprocedural propagation
  - I/O times dominate (loading/unloading large tables)
- Under investigation: caching tables on machines of the cluster and using PVM communication mechanism (faster than concurrent database access)

	Size (KLOC)	Max Size Analyzed	Precision	Analysis Time (hours)
MPF	140	140	80%	1.5
DS1	280	280	80%	2.5
MER	550	550	80%	20

C Global Surveyor

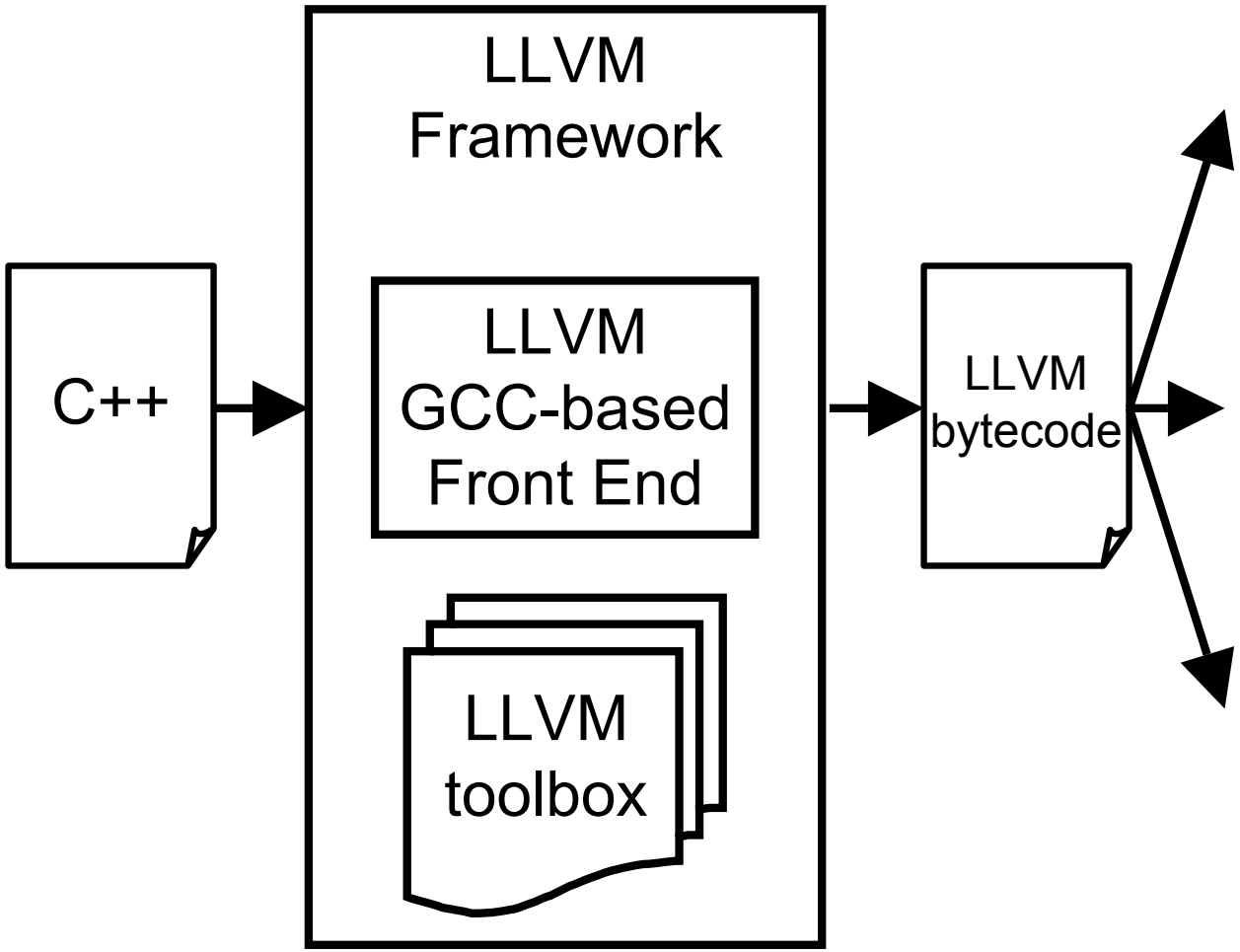
- Mars & Solar System Exploration (JPL)
  - MER
  - MSL
- Manned space missions: International Space Station & Shuttle
  - Urine Processing Assembly (20KLOC)
  - Material Science Research Rack (82KLOC)
  - Advanced Video Guidance System (12KLOC)
  - Space Shuttle Main Engine Controller(?)
  - Biological Research Project Rack Interface Controller (40KLOC)
  - Centrifuge Rack Interface Controller (40KLOC)
- Independent Verification & Validation Center

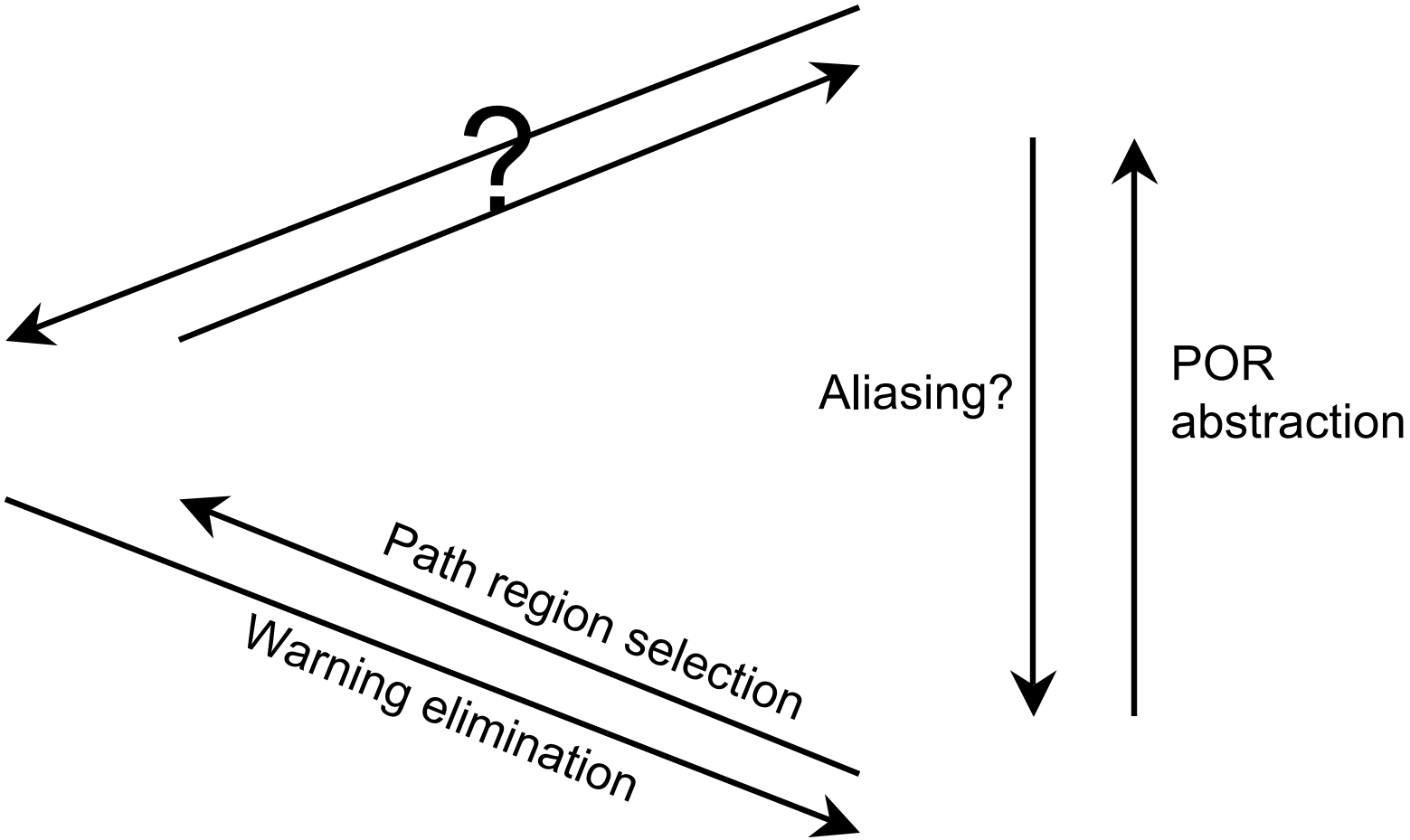
Done without daily expert help

- Static analyzer for finding runtime errors in C programs
  - Out-of-bound array accesses
  - De-referencing null pointers
  - Tested on MPF, DS1, and ISS flight software systems
- Developed (20 KLoc of C) at NASA Ames in ASE group
  - A. Venet: no longer working at NASA
  - G. Brat: brat@email.arc.nasa.gov
  - S. Thompson: thompson@email.arc.nasa.gov
- Runs on Linux and Solaris platforms
  - RedHat Linux 2.4
- Analysis can be distributed over several CPUs
  - Using PVM distribution system
- Results available using SQL queries
  - To the PostgreSQL database
  - Graphical user interface

- Need to move to analyzing C++
  - C is the legacy language
  - New development (CEV, CLV) is in C++
- C++ is a complicated language
  - Dynamic allocation
  - Virtual functions
  - Object-oriented
  - No thread standard package
- Our strategy
  - Develop more than just static analysis tools
  - Based them on the same language compilation framework

LLVM Compilation and Analysis Framework
- Open source
- Used by Apple for commercial products


LLVM
Framework
C++
LLVM
GCC-based
Front End
LLVM
toolbox
LLVM
bytecode


?
Aliasing?
POR
abstraction
Path region selection
Warning elimination

- Static analysis is useful for NASA software
  - Certifies the absence of errors
  - Does not require testing/simulation environment
- Static analysis is becoming practical
  - Scales to large software (e.g., MER)
  - Number of false positives is greatly reduced
  - Analysis times are less than a day even for large software
- CGS (developed at NASA Ames Research Center)
  - Catches pointer manipulation errors in embedded C programs
  - Is applicable to large flight software
- We are working on a suite of C++ analysis tools
  - Model checker
  - Symbolic execution
  - Static analysis